Angular Fundamentals
Module 2 - Databinding

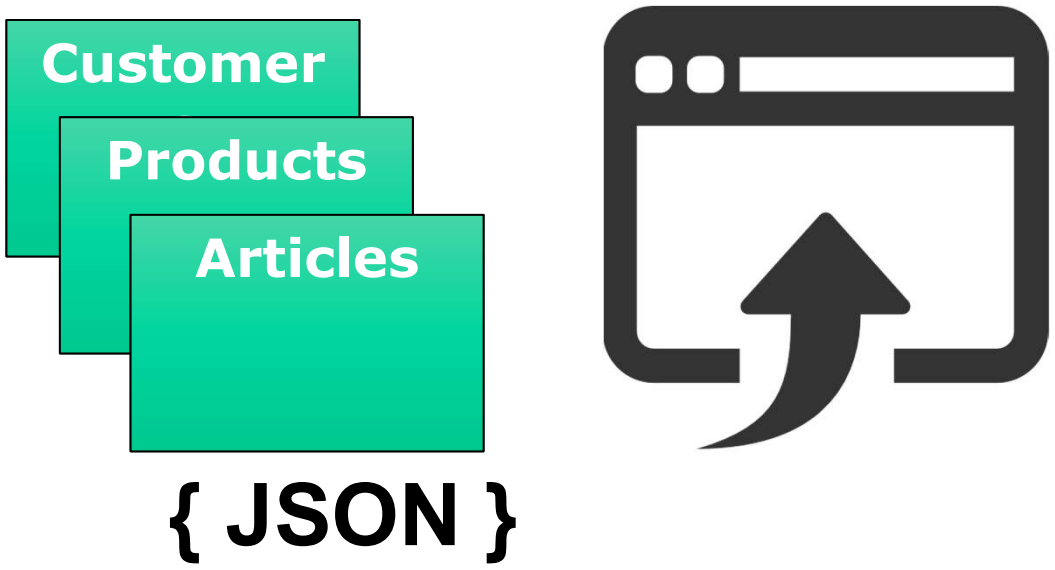
Peter Kassenaar –
info@kassenaar.com

Hoofdstuk 3

p. 72 en verder

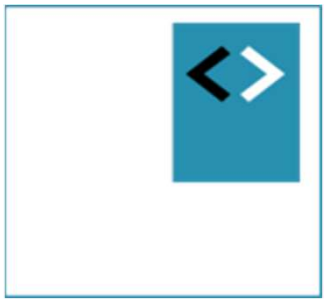
Wat is databinding

- Gegevens (data) tonen in de user interface
- Data afkomstig uit:
  - Controller / class
  - Database
  - User input
  - Andere systemen



Declaratieve syntaxis

- Vier manieren voor databinding in HTML-views/templates.
    1. Simple data binding
    2. Event binding
    3. One-way data binding
    4. Two-way data binding



1. Simple Data binding

Class-properties binden in de template

1. Simple data binding syntax

Ongewijzigd ten opzichte van Angular 1. Dus nog steeds dubbele accolades:

```
<div>Stad: {{ city }}</div>

<div>Voornaam: {{ person.firstname }}</div>
```

Altijd: samenwerking met component/class

```
import {Component} from '@angular/core';
@Component({
    selector: 'hello-world',
    template: `<h1>Hello Angular</h1>
        <h2>Mijn naam is : {{ name }}</h2>
        <h2>Mijn favoriete stad is : {{ city }}</h2>
    `
})
export class AppComponent {
    name = 'Peter Kassenaar';
    city = 'Groningen'
}
```

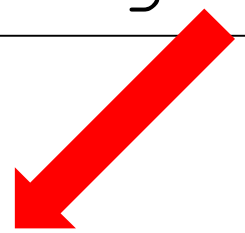
Of: properties later instellen

```
export class AppComponent {
    name: string;
    city: string;

    constructor() {
        this.name = '…';
        this.city = '…';
    }
     ngOnInit() {
        this.name = 'Peter Kassenaar';
        this.city = 'Groningen';
    }
}
```

BEST PRACTICE:

use `ngOnInit()`



Binden via een lus: `*ngFor`

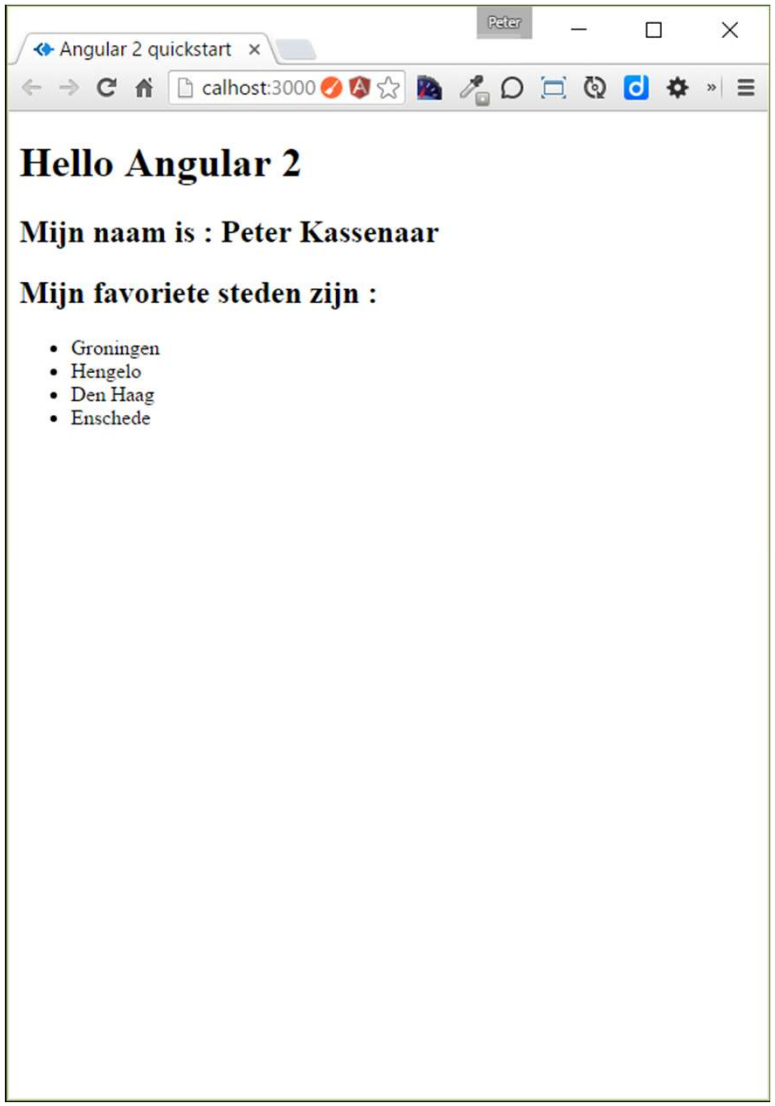
Template:

```
<h2>Mijn favoriete steden  zijn :</h2>
<ul>
   <li *ngFor="let city of cities">{{ city }}</li>
</ul>
```

Class:

```
// Class met properties, array met cities
export class AppComponent {
   name:string;
   cities:string[];

   ngOnInit() {
      this.name   = 'Peter Kassenaar';
      this.cities = ['Groningen', 'Hengelo', 'Den Haag', 'Enschede']
   }
}
```



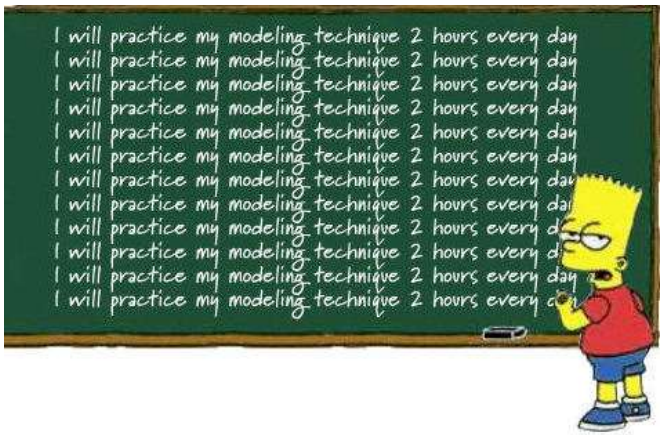
Meer info:

https://angular.io/guide/displaying-data

Checkpoint

- Simple data binding `{{ … }}`
- Properties van de class worden gebonden
- Lussen via `*ngFor`
- Data staat in een array
- Oefening `2a)` en `2b)`

Oefening….



Model maken (als in: MVC)

Class met properties die wordt geëxporteerd:

```
export class City{
   constructor(
      public id: number,
      public name: string,
      public province: string,
   ){ }
}
```

Let op de shorthand notatie bij `public id : number` :

1. Maakt lokale parameter
2. Maakt publieke parameter met zelfde naam
3. Initaliseert parameter bij instantiering van de class met `new`

Model gebruiken

1. Model-class importeren

```
import {City} from './city.model'
```

2. Component aanpassen

```
export class AppComponent {
      name   = 'Peter Kassenaar';
      cities =[
         new City(1, 'Groningen', 'Groningen'),
         new City(2, 'Hengelo', 'Overijssel'),
         new City(3, 'Den Haag', 'Zuid-Holland'),
         new City(4, 'Enschede', 'Overijssel'),
      ]
}
```

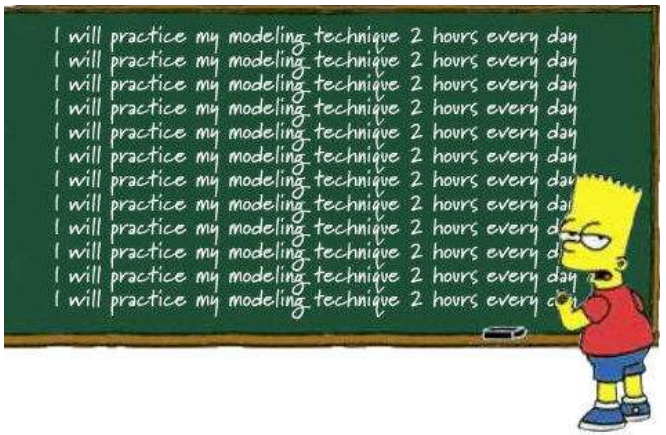
3. View aanpassen

```
<li *ngFor="let city of cities">{{ city.id}} - {{ city.name }}</li>
```

Checkpoint

- Model maken en gebruiken: `Class` of `interface`
- Denk aan de juiste `import`-statements
- Best practice; plaats je class (of interface) in de map `/shared`.
- Oefening `2c)`

Oefening….



Voorwaardelijk tonen met `*ngIf`

Gebruik de directive `*ngIf` (let op het sterretje!)

```
<h2 *ngIf="cities.length > 3">Jij hebt veel favoriete steden!</h2>
```

Externe templates

Als je niet van inline HTML houdt:

```
@Component({
   selector   : 'hello-world',
   templateUrl: './app.component.html'
})
```

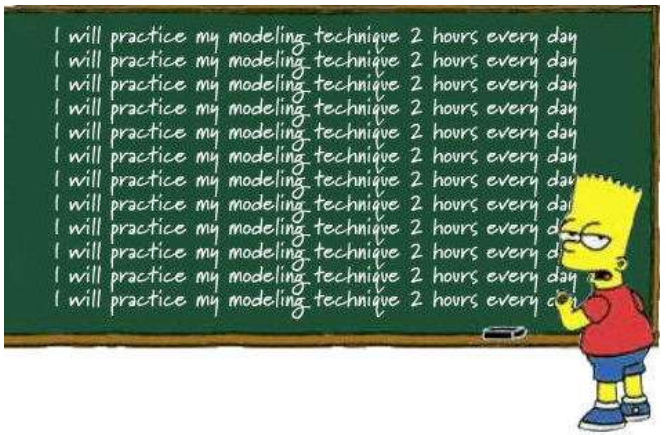
Bestand `app.component.html`

```
<!-- HTML in externe template -->
<h1>Hello Angular 2</h1>
<p>Dit is een externe template</p>
<h2>Mijn naam is : {{ name }}</h2>
<h2>Mijn favoriete steden  zijn :</h2>
…
```

Checkpoint

- Simple data binding `{{ … }}`
- Gebruik bij voorkeur een Model (`class` of `interface`)
- Lussen en voorwaardelijke statement via `*ngFor` en `*ngIf`
- Eventueel : externe HTML-templates
- Oefening `2c)` en `2d)`

Oefening….



2. User input en event binding

Reageren op mouse, keyboard, hyperlinks en meer

Hoofdstuk 4

p. 98 en verder

Event binding syntaxis

Gebruik ronde haken voor events:

Angular 1:

```
<div ng-click="handleClick()">…</div>
```

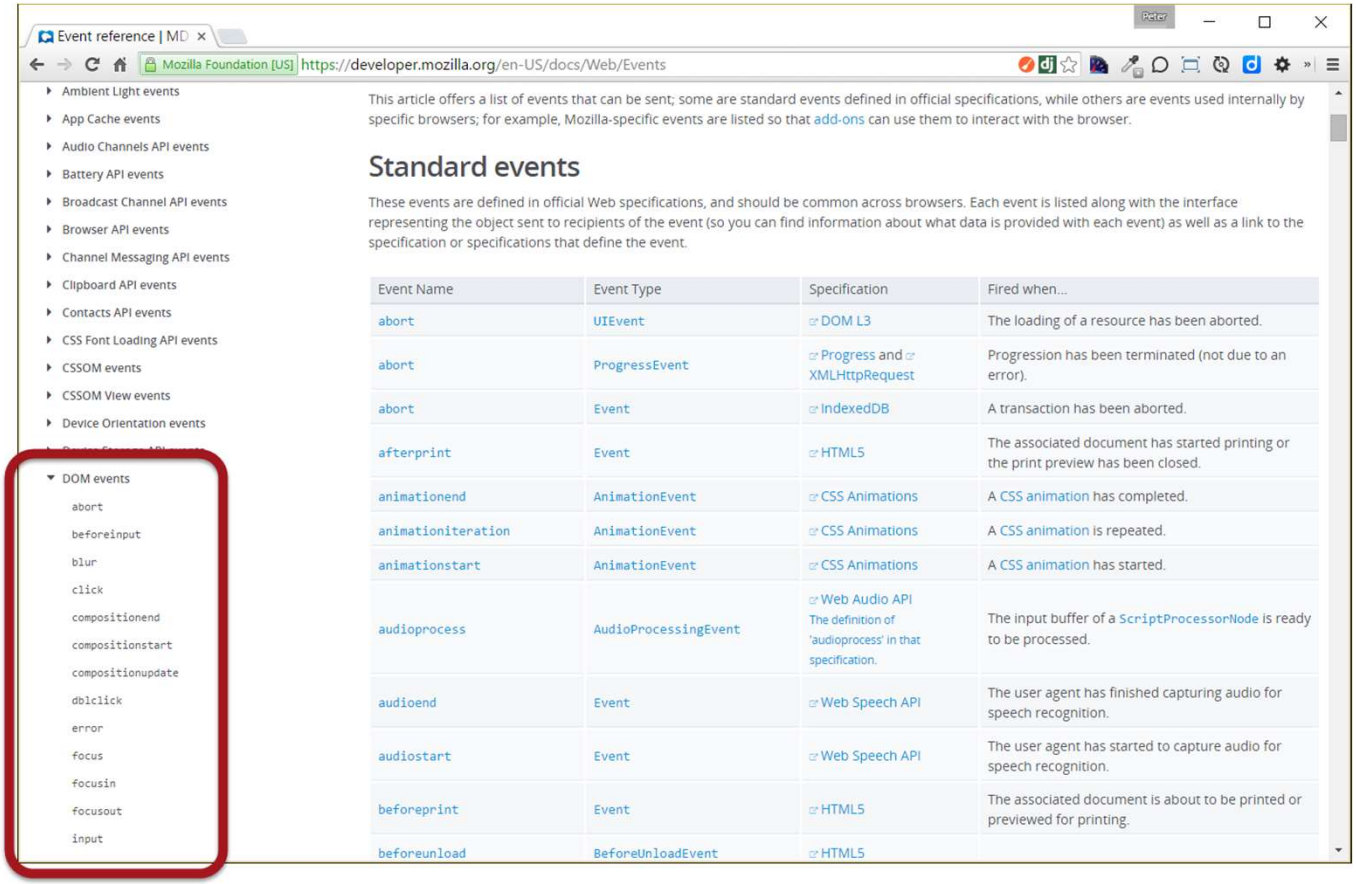
Angular 2:

```
<div (click)="handleClick()">…</div>

<input (blur)="onBlur()">…</div>
```

DOM-events

Angular2+ kan naar *elk* DOM-event luisteren, zonder dat er een aparte directive voor nodig is:



https://developer.mozilla.org/en-US/docs/Web/Events

2a. Niet-DOM events binden

- Niet-DOM events binden: `@HostListener()`
- Luister naar events op het `window`-object, decoreer Event Listener functie.
- Doorgeven van `$event` is optioneel
- Bijvoorbeeld:

```
// Decorator voor capture van non-DOM events
@HostListener('window:offline', ['$event'])
onOffline(event) {
    this.msg = 'We zijn offline!';
    console.log('we zijn nu offline ==>', event);
}
```

```
// Luisteren naar niet-DOM events: gebruik
// de decorator @HostListener()
@HostListener('window:offline',['$event'])// $event is optioneel
onOffline(e) {
    console.log(e);
    this.msg = 'We zijn offline!';
    console.log('We zijn offline!');
}

@HostListener('window:online')
onOnline() {
    this.msg = 'We zijn weer online! Ga synchronisen';
    console.log('We zijn online!');
}
```

Voorbeeld event binding

HTML

```
<!-- Event binding voor een button -->
<button class="btn btn-success"
      (click)="btnClick()">Ik ben een button</button>
```

Class

```
export class AppComponent {
   …
   counter: number =0;

   btnClick(){
      alert('Je hebt '+ ++this.counter +' keer geklikt');
   }
}
```

- Veel editors geven intellisense voor de beschikbare events

a. Event-parameters: $event

HTML

```
<input type="text" class="input-lg" placeholder="Plaatsnaam..."
       (keyup)="onKeyUp($event)"><br>

<p>{{ txtKeyUp}}</p>
```

Class

```
// 2. Binden aan keyUp-event in de textbox
onKeyUp(event:any){
   this.txtKeyUp = event.target.value + ' - ';
}
```

b. Event-parameters: local template variable

Declareer *local template variable* met # → Het hele element wordt doorgegeven aan de component

HTML

```
<input type="text" class="input-lg" placeholder="Plaatsnaam..."
      #txtCity (keyup.enter)="betterKeyUp(txtCity)">
<h3>{{ txtCity.value }}</h3>
```

Class:

```
 // 3. Binden aan keyUp-event via local template variable
betterKeyUp(txtCity:HTMLInputElement){
   //... Doe iets met txtCity…
}
```

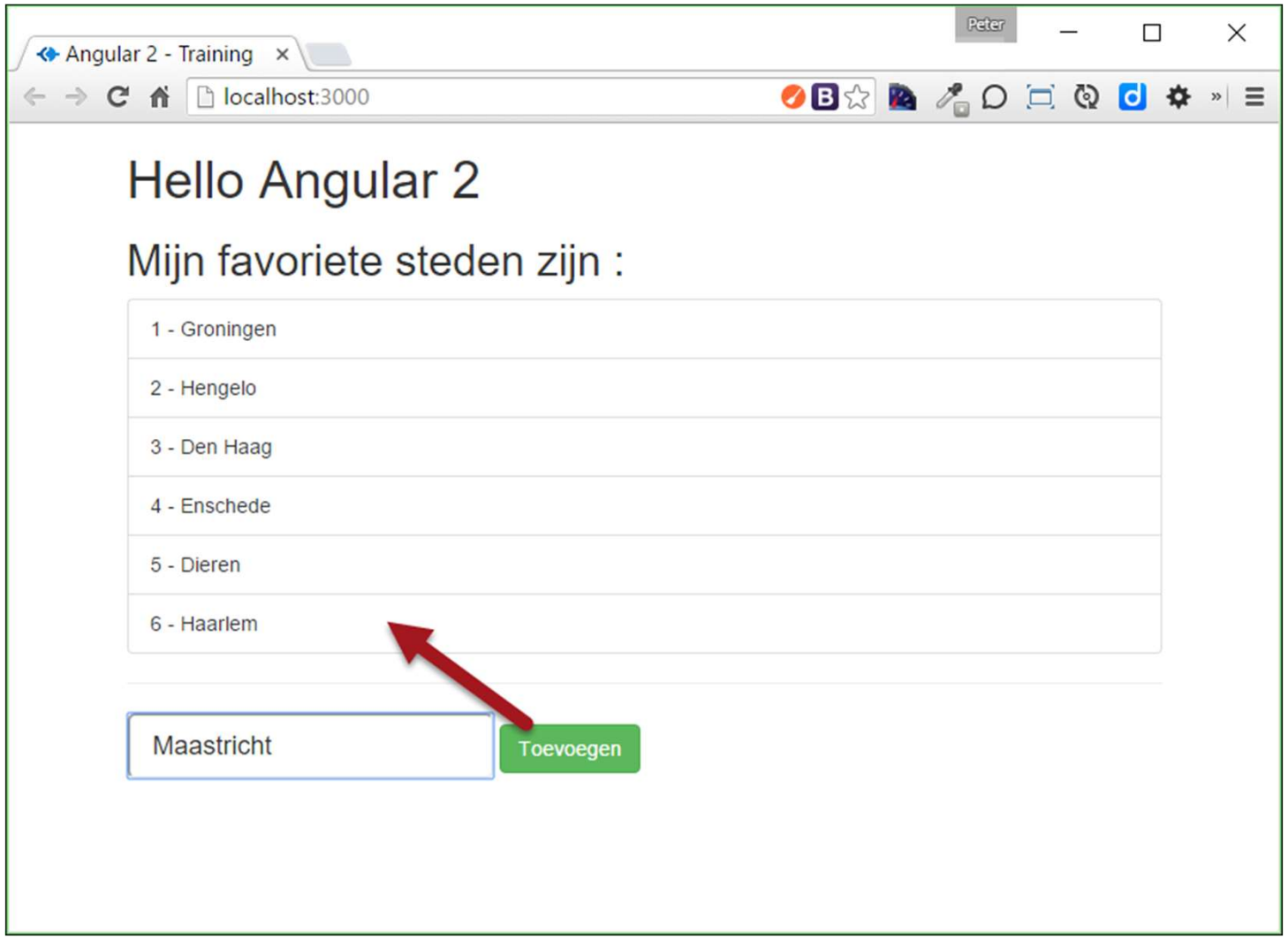
Putting it all together…

HTML

```
<input type="text" class="input-lg" placeholder="Plaatsnaam..." #txtCity>
<button class="btn btn-success"
        (click)="addCity(txtCity)">Toevoegen
</button>
```

Class

```
export class AppComponent {
   // Properties voor de component/class
   …
   addCity(txtCity:HTMLInputElement) {
      let newID   = this.cities.length + 1;
      let newCity = new City(newID, txtCity.value, 'Onbekend');
      this.cities.push(newCity);
      txtCity.value = '';
   }
}
```

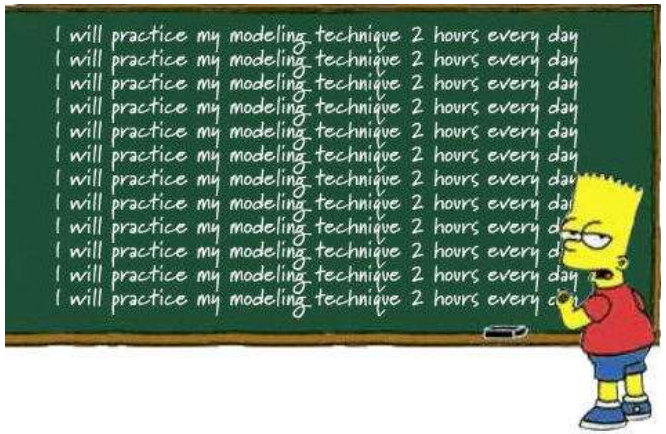


Verder lezen/meer informatie: https://angular.io/docs/ts/latest/guide/user-input.html

Checkpoint

- Event binding wordt aangegeven met `(eventName)="…"`
- Events worden afgehandeld door een event handler-functie in de component
- Gebruik `$event` om het complete, ruwe browserevent door te geven aan de controller
- Gebruik # voor *local template variable*
- Op deze manier zijn eenvoudige client-sided CRUD-operations te realiseren.
- Oefening `3b), 3c), 3d), 3e)`

Oefening….



3. Attribute & property binding

Eigenschappen binden aan HTML-attributen en DOM-properties

Attribute binding syntaxis

Rechtstreeks binden aan properties van HTML-elementen.
Ook wel: *one-way binding.*
Gebruik blokhaken syntaxis

Angular 1:

```
<div ng-hide="true|false">…</div>
```

Angular 2:

```
<div [hidden]="true">…</div>
```

Of :

```
<div [hidden]="person.hasEmail">…</div>
<div [style.background-color]="myBgColor">…</div>
```

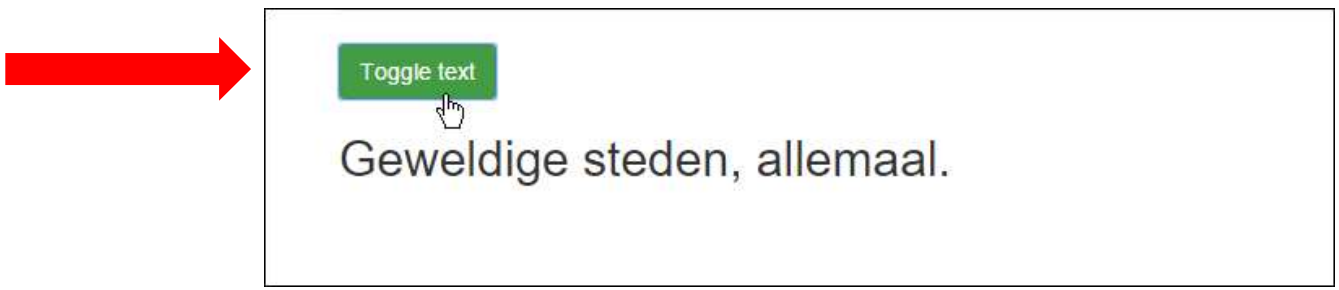
Voorbeeld attribute binding

HTML

```
<!-- Attribute binding -->
<button class="btn btn-success" (click)="toggleText()">Toggle text</button>
<h2 [hidden]="textVisible">Geweldige steden, allemaal.</h2>
```

Class

```
// attribuut toggelen: tekst zichtbaar/onzichtbaar maken.
toggleText(){
   this.textVisible = !this.textVisible;
}
```



Bijvoorbeeld…

HTML

```
<li *ngFor="let city of cities" class="list-group-item"
   (click)="updateCity(city)">
   {{ city.id}} - {{ city.name }}
</li>
```

Class

```
export class AppComponent {
   // …
   currentCity:City    = null;
   cityPhoto:string    = '';

   // Geselecteerde city updaten in de ui. Nieuw : ES6 String interpolation
   updateCity(city:City) {
      this.currentCity = city;
      this.cityPhoto   = `img/${this.currentCity.name}.jpg`;
   }
}
```

Demo:

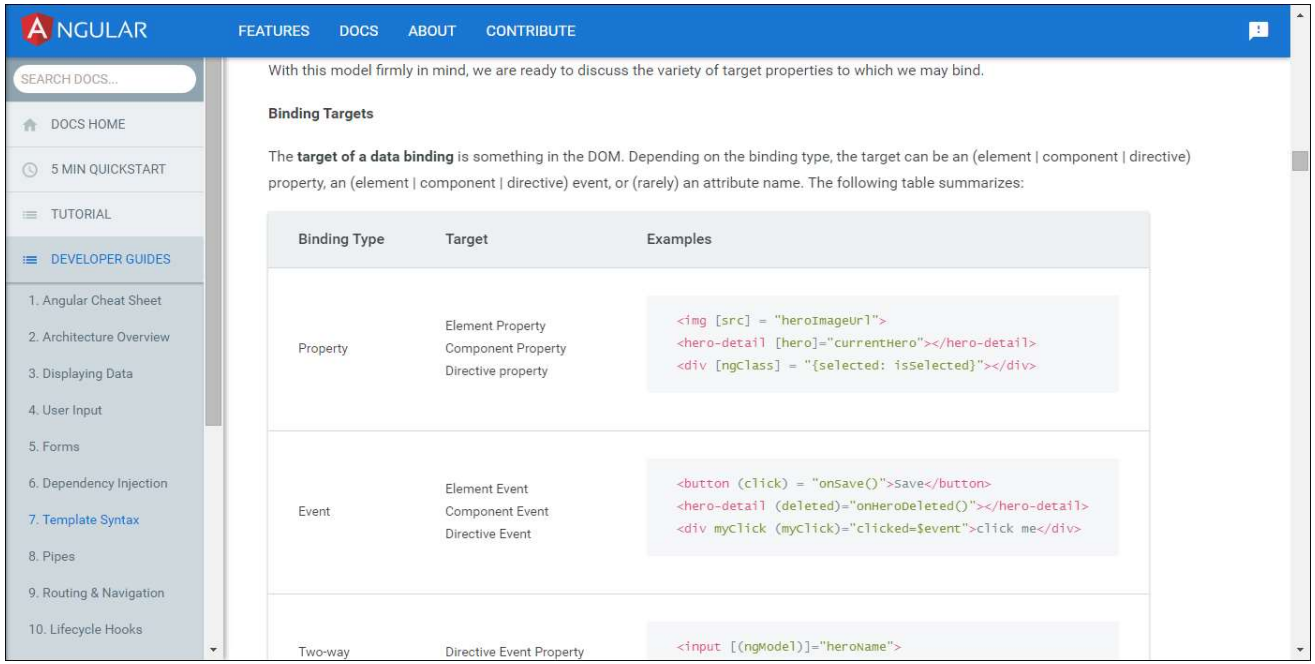
`..\103-attributebinding\app\app-02.html` en

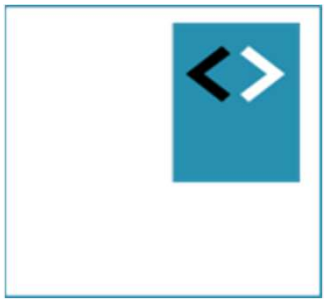
`..\app-02.component.ts`

Meer informatie: https://angular.io/docs/ts/latest/guide/template-syntax.html#!#property-binding

Meer binding-opties

- Attribute binding en DOM-property binding
- Class binding
- Style binding
- https://angular.io/docs/ts/latest/guide/template-syntax.html


ANGULAR
FEATURES
DOCS
ABOUT
CONTRIBUTE
SEARCH DOCS...
DOCS HOME
5 MIN QUICKSTART
TUTORIAL
DEVELOPER GUIDES
1. Angular Cheat Sheet
2. Architecture Overview
3. Displaying Data
4. User Input
5. Forms
6. Dependency Injection
7. Template Syntax
8. Pipes
9. Routing & Navigation
10. Lifecycle Hooks
With this model firmly in mind, we are ready to discuss the variety of target properties to which we may bind.
Binding Targets
The target of a data binding is something in the DOM. Depending on the binding type, the target can be an (element | component | directive) property, an (element | component | directive) event, or (rarely) an attribute name. The following table summarizes:
Binding Type
Target
Examples
Property
Element Property
Component Property
Directive property
<img [src] = "heroImageUrl">
<hero-detail [hero]="currentHero"></hero-detail>
<div [ngClass] = "{selected: isSelected}">
Event
Element Event
Component Event
Directive Event
<button (click) = "onSave()">Save</button>
<hero-detail (deleted)="onHeroDeleted()"></hero-detail>
<div myClick (myClick)="clicked=$event">click me
Two-way
Directive Event Property
<input [(ngModel)]="heroName">



4. Two-way binding

User interface en logica gelijktijdig updaten

Two way binding syntaxis

Is een tijdje weg geweest uit Angular 2, maar op veler verzoek toch teruggekeerd

Angular 1:

```
<input ng-model="person.firstName" />
```

Angular 2: de notatie is een beetje bizar:

```
<input [(ngModel)]="person.firstName" />
```

[(ngModel)] gebruiken

HTML

```
<input type="text" class="input-lg" [(ngModel)]="newCity" />
<h2>{{ newCity }}</h2>
```

Dat is shorthand-notatie voor:

```
<!-- Two-way binding met uitgebreide syntaxis-->
<input type="text" class="input-lg"
       [value]="newCityExtended"
       (input)="newCityExtended = $event.target.value" />
<h2>{{ newCityExtended }}</h2>
```

FormsModule importeren

- Vroeger maakte de Formulier-functionaliteit standaard deel uit van Angular.
- Nu niet meer – apart importeren in `app.module.ts`!

- `import {FormsModule} from "@angular/forms";`
- …
- `imports      : [BrowserModule, FormsModule],`

Dus: data doorgeven van View → Controller

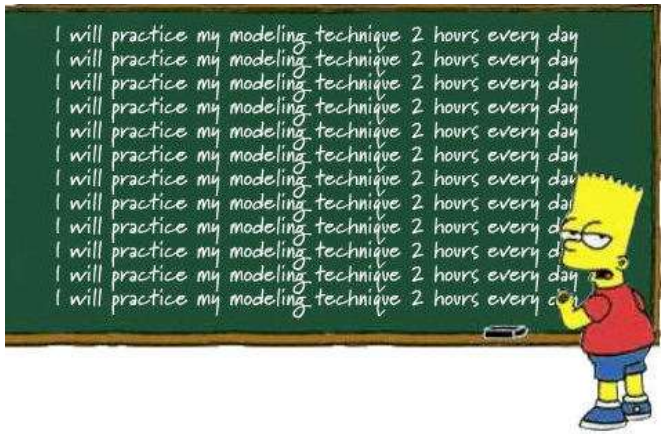
1. Using `$event`
2. Using a Local Template Variabele `#NameVar`
3. Using `[(ngModel)]` (to be used in simple situations, mostly not on complex forms)
4. HostBinding/`@HostListener` (via @-decorators)
5. Decorator `@ViewChild()`

Checkpoint

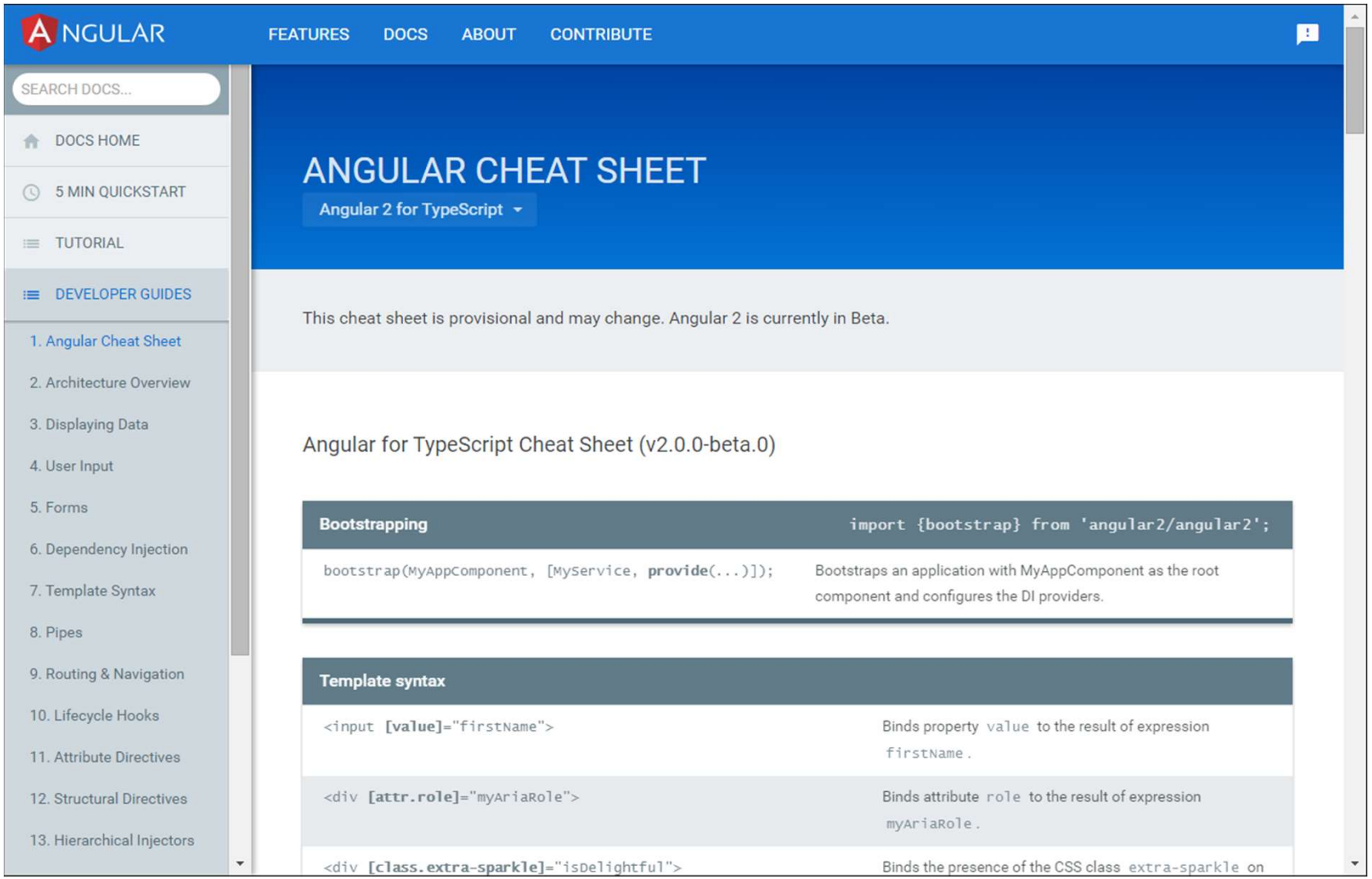
- Attribute binding wordt aangegeven met

  `[attributeName]="…"`
- Op deze manier maak je attributen van HTML-tags dynamisch
- Aan de rechterkant plaats je een variabele van de component
- Via `[(ngModel)]` kun je in Angular two-way binding gebruiken
- Oefening `4a), 4b), 4c)`

Oefening….



Binding cheat sheet



https://angular.io/docs/ts/latest/guide/cheatsheet.html

Ingebouwde directives

Veel directives konden vervallen door de nieuwe syntaxis. Er zijn er nog maar weinig over.

Directives die het DOM manipuleren: herkenbaar aan sterretje/asterisk

```
<div *ngFor="person of Persons">…</div>

<div *ngIf="showDiv">…</div>

<div [ngClass]="setClasses()">…</div>

<div [ngStyle]="setStyles()">…</div>
```

Samenvatting…

- Databinding is in Angular vernieuwd
- Leer werken met de nieuwe notatie voor DOM- en Attribute binding, event binding en two-way binding
- Pas altijd de Component en de bijbehorende View aan.
- Veel concepten komen overeen, de uitwerking is anders dan in Vue, React en Angular 1

Samenvatting

- Vier manieren voor databinding in HTML-views/templates.
    1. Simple data binding met {{ … }}
    2. Event binding met (…)
    3. One-way data binding met […]
    4. Two-way data binding met [(ngModel)]